UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/693,630	10/23/2003	Sriram Subramanian	13768.783.230	1530

47973 7590 03/20/2008
WORKMAN NYDEGGER/MICROSOFT
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UT 84111

EXAMINER	
AMIN, JWALANT B	

ART UNIT	PAPER NUMBER
2628	

MAIL DATE	DELIVERY MODE
03/20/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/693,630	SUBRAMANIAN ET AL.	
	Examiner	Art Unit	
	JWALANT AMIN	2628	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE _3_ MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on _17 October 2007_.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) _1-35 and 68_ is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) _1-35 and 68_ is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

Response to Arguments

1. Applicant's arguments with respect to claims 1 and 68 have been considered but are moot in view of the new ground(s) of rejection.

2. Regarding claims 1 and 68, the applicant argues "…the cited art fails to teach or suggest an receiving a function call through an API providing access to functionality of a media integration layer which comprises a plurality of types of objects including a plurality of VisualManager objects, each VisualManager object connecting a Visual Tree to a particular medium, each VisualManager object having a relationship with a window in which graphic data is output, and each VisualManager managing the rendering process to the particular medium, a Visual Tree to a render target which is a particular medium, the Visual Tree comprising a plurality of Visuals, and each Visual providing parent visual access, child visual collection, clipping, opacity, blendmode, transform, hit testing, and bounding box services, and rendering, by a VisualRenderer, the Visual Tree to the particular medium and causing a change in a graphics display of the particular medium in response to the modification of data in the scene graph" (see pg. 3 of applicant's remarks).

3. The examiner interprets that Beda in view of Louveaux teaches these limitations. Please refer to the rejection of claim 1 below for details.

Claim Rejections - 35 USC § 101

4. 35 U.S.C. 101 reads as follows:

> Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

5. Claim 68 is rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter.

6. Regarding claim 68, the language of the claim raise questions as to whether the claim is directed merely to an abstract idea that is not tied to a technological art, environment or machine which would result in a practical application producing a concrete, useful, and tangible result to form the basis of statutory subject matter under 35 U.S.C. 101. Specifically, the computer-readable media, as disclosed in claim 68 and described on page 13-14 of the specification, in the context of this disclosure covers carrier waves, which are not a Manufacture within the meaning of 101. See MPEP 2106.

Claim Rejections - 35 USC § 103

7. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

8. Claims 1 and 68 are rejected under 35 U.S.C. 103(a) as being unpatentable over Beda et al. (US 2003/0076329; hereinafter Beda), and further in view of Louveaux et al (US 7102651; hereinafter Louveaux).

9. Regarding claims 1 and 68, Beda teaches a computer-implemented method for arranging vector graphics data for processing into an output (Beda [0072-0073] teaches storing vector graphics in a "primitive list" or primitive container), the method comprising:

- Receiving a function call via an application programming interface (API) of a media integration layer (MIL), (MIL has API - Beda teaches Media Integration Layer in [0039-0041], that has APIs by which higher level programs (eg user GUI in OS) access MIL [0040] to render output graphics [0011,0013,0017], where such API has functions [0070,0111,0113] that are called (Abstract, [0014]), and the API receives such calls [0017,0039-0041])

- The API providing access to the functionality of the MIL ([0040]; higher level program code accesses the MIL architecture via a set of APIs).

- MIL comprising a plurality of types of objects (display tree filled with objects, [0044]; MIL has multiple layers [0041], where the 'high-level' and 'low-level engines comprise 'layers' (e.g. layer=level) [0043,0046,0048], which generates a plurality of objects in tree form ([0044,0057]; Figure 4, element 300; Figure 5, element 500; Figure 2, block 206 generates the tree [0043]), the objects including a plurality of VisualManager objects (display manger; display manager comprises a hosting object that represents an instance of the high-level compositor and animator, [0066]; there may be a suitable instance of a high-level compositor and animator to process the output of an application for each type of abstract device [0050]), each VisualManager object connecting a Visual Tree to a particular medium (fig. 4; display manager

represents an instance of high-level composition and animation engine, which builds a display tree and traverses the tree creating rendering instructions to be passed to a graphics sub-system or a printer or to a lower-level terminal transport server, fig. 2, [0043-0045], [0066]), each VisualManager object having a relationship with a window in which graphic data is output ([0066]), and each VisualManager managing the rendering process to the particular medium (display manager comprises a hosting object that represents an instance of the high-level compositor and animator, [0066]; there may be a suitable instance of a high-level compositor and animator to process the output of an application for each type of abstract device [0050]; display manager represents an instance of high-level composition and animation engine, which builds a display tree and traverses the tree creating rendering instructions to be passed to low-level compositing and animation engine or a printer or to a lower-level terminal transport server, [0043-0045], [0066]),

- connecting, by a VisualManager, a Visual Tree to a render target which is a particular medium (fig. 4; display manager represents an instance of high-level composition and animation engine, which builds a display tree and traverses the tree creating rendering instructions to be passed to low-level compositing and animation engine or a printer or to a lower-level terminal transport server, [0043-0045], [0066]; The system has a display manager object 420 (Figure 4) that manages the tree [0066] by utilizing a pointer to the root container of the tree, additionally as in Figure 6, element 604 [0091]), the Visual Tree comprising a plurality of Visuals (fig. 7 shows a visual tree containing a plurality of visuals including root node 720, which is a special visual and it's

children nodes that are top level visuals, ([0091-0105]), each Visual providing parent visual access (fig. 7 shows root node 720 provides access to parent visuals TLV_1-TLV_3, [0091]), child visual collection (fig. 7 shows parent visuals TLV_1-TLV_3 has a collection of child visual nodes; adornments 730_1-730_3 visuals are children of top level visual to which they belong; [0100-0105]), clipping ([0066]), opacity ([0109]), blendmode, transform ([0066], [0085]), hit testing ([0066], [0080-0084]), and bounding box services ([0077-0078], [0118];

- traversing the Visual Tree ([0044], [0101]);

- rendering, by a VisualRenderer, the Visual Tree to the particular medium (fig. 2, fig. 3, fig. 4, [0043-0045], [0100-0101]; display manager represents an instance of high-level composition and animation engine, which builds a display tree and traverses the tree creating rendering instructions to be passed to a graphics sub-system or a printer or to a lower-level terminal transport server);

- Causing a change in a graphics display of the particular medium in response to the modification of data in the scene graph. ([0011-0012], [0016]).

Although Beda teaches the limitations as stated above, Beda does not explicitly teach a Visual providing opacity and blendmode properties. However, Louveaux teaches that all levels of the graphical objects (Visuals) can be specified with opacities and blending modes (col. 1 lines 32-34 and col. 2 lines 39-41). Therefore, it would have been obvious to one of ordinary skill in art at the time of present invention to provide a Visual with properties like opacity and blending mode as taught by Louveaux and use it into the method of Beda because by providing a visual with opacity and blending mode,

the user does not have to worry about whether a given piece of art already contains transparency attributes (col. 2 lines 39-44).

10. Claims 2-27 and 30-35 are rejected under 35 USC 103(a) as unpatentable over Beda and Louveaux, and further in view of Steele (US 2004/0110490 A1), which incorporates the SVG specification by reference. The motivation and rationale for combination of Steele with the above references is provided in claim 2. That motivation and rationale is herein EXPLICITLY INCORPORATED INTO ALL OTHER CLAIMS BELOW BY REFERENCE.

11. Regarding claim 2, Beda and Louveaux do not implicitly teach causing data in the scene graph to be modified comprises causing initialization of a new instance of a non-drawing visual class. However, Steele implicitly teaches this limitation, that SVG data is decomposed into scene graphs, a.k.a. trees, (see Figure 7), and again - whenever new visual elements enter the scene, new subgroups are instantiated, which prima facie (see SVG specification, section 9) are elements that compose visual objects, which therefore are new instances of a visual class as recited above, since a class as recited by applicant is comparable to the basic 'shapes' in SVG (applicant's specification clearly uses it - 23:1-20 where applicant's invention adopts all classes and shapes from SVG) and thusly meets the recited limitation, wherein these are non-drawing visual subclasses.

Steele clearly teaches the use of a SVG DOM as element 305 in Figure 3, where that intermediate format is then transferred to the BF object model 315, where this

clearly represents a scene graph (see Figure 6), where vector elements 610 and behavior elements 620 exist, where these form scene graph - see Figure 7, which shows the visual elements 610 as a visual graph [0051-0058], see 0051-0053 and 0057-0058 specifically. The other elements are represented as a sequence graph 800. These graphs are matched to each other, as in elements 920 and 930, where the original SVG is shown in Figure 910. Clearly the Sequence Graph portion manipulates the visual object layer, such that manipulating the Visual Graph to change locations does the animation of an object, create new objects, etc. SVG can supply animation commands that discuss where the objects are located and the like with attribute and animate commands. Finally, it is clear that the SVG objects shown in Steele have properties associated with them such as size and/or animation information (as an example, see Figure 910).

It is well known in the art and shown by Steele in the referenced paragraphs above that SVG objects can have commands such as animation attached to them. Obviously such SVG intermediate data structures (e.g. SVG DOM tree, or SVG DOM in the Steele drawings) have to be translated to a lower, implementation level for actual drawing on the scene. The visibility and/or presence of such objects in the lower level (e.g. scene graph) are obviously controlled by their properties (Steele code as an example)(where such implementations are in Java [0095, 0152]).

It would therefore have been obvious to one of ordinary skill in the art at the time the invention was made to modify Beda in view of Louveaux with the SVG

implementations of Steele because this allows for more efficient implementation for looping behavior and fewer modifications to the scene graph [0061-0068].

12. As to claim 3,

The method of claim 2 wherein causing data in the scene graph to be modified comprises receiving a function call via an interface corresponding to a transform associated with the visual.

Beda and Louveaux do not expressly teach, but Steele teaches this limitation, as he discloses rotations in [0053] and further states that rotations and other transformations can be applied to an entire tree of objects, e.g. Fig. 7, and further [0088] that any visual element or object can modified. Such modifications prima facie must associate code with a suitable / desired transform (e.g. scaling, rotation, et cetera [0053]), as that is the only way either a hierarchy of nodes (e.g. Fig. 7) or single node could be scaled. (Further, note that since this is performed by software, prima facie 'code' that is software elements, would be invoked to perform any recited task.)

As to claim 4,

The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a function to initialize a new instance of a drawing visual class.

Beda and Louveaux do not expressly teach, but reference Steele teaches this limitation, as for example he teaches the insertion of unique identifiers into media streams [0106], and further [0088] that any visual element or object can modified. Such modifications and insertions prima facie must associate code with a suitable / desired

insertion as that is the only way either a hierarchy of nodes (e.g. Fig. 7) or single nodes could be logically inserted.

For the second case, if the definition of context is the data associated with a specific element - e.g. the details of the element, its location, color, et cetera, these attributes are inherent in SVG elements as set forth in the rejections above, e.g. sections 11.1, 9.1-9.7, et cetera. Further, Steele teaches the same in Figure 7, where each element has certain properties that would be a drawing context, in the sense that each visual element has those properties associated with it [Steele 0052-0056 and 0059-0061].

As to claim 5

The method of claim 4 further comprising, causing drawing context to be returned, the drawing context providing a mechanism for rendering into the drawing visual,

Beda and Louveaux do not expressly teach, but reference Steele teaches this limitation, as for example he teaches the retrieval of device context in [0101]. Clearly, the device receives information based on its device context, which clearly is associated with the drawing context, as the two are one and the same in this case. Steele teaches rendering in [0007 and 0011-0012]. The drawing context per se is incorporated into the data structures of Steele (see Figure 7). It further would have been obvious to modify the system of Louveaux to utilize a device specific context so as to optimize data streamed to that device for purposes of minimizing memory consumption (a large problem pointed out by Steele [0007]).

As to claim 6,

The method of claim 1 further comprising, receiving brush data in association with the function call, and wherein causing data in the scene graph to be modified comprises invoking a brush function to modify a data structure in the scene graph data structure such that when a frame is rendered from the scene graph, an area will be filled with visible data corresponding to the brush data.

Beda and Louveaux do not expressly teach, but reference Steele does teach this limitation by the use of SVG graphics. Turning to the SVG (, section 11 titled 'Painting: Filling, Stroking, and Marker Symbols', specifically section 11.1, 'With SVG, you can paint (e.g. stroke or fill) with: ..." and then proceeds to list several. The term 'brush data' is clearly analogous to the 'paint' operation in SVG with comparable data. Given that SVG allows (from section 11.1) a single color, a solid color (with or without opacity), a gradient, a pattern (vector or image), and custom patterns, clearly each visible element clearly has such data associated with it (see section 11.2 in its entirety, 11.7 for specific properties, section 11.8 for how painting properties can be inherited, which prima facie justifies the position that element have intrinsic painting properties, i.e. brush data as set forth above. Further, note that since this is performed by software, prima facie 'code' that is software elements, would be invoked to perform any recited task, and SVG data element prima facie and inherently possess paint data as set forth by the SVG specification above. The SVG standard inherently handles these paint limitations. SVG inherently handles these paint limitations.

As to claim 7,

The method of claim 6 wherein the brush data comprises receiving data corresponding to a solid color.

Beda and Louveaux do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets forth that a user can paint with a solid color with opacity, thus meeting this limitation.

As to claim 8,

The method of claim 6 wherein receiving brush data comprises receiving data corresponding to a linear gradient brush and a stop collection comprising at least one stop.

Reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets forth that a user can paint with a gradient that can be linear. Further, sections 11.7.1 and 11.7.2 of the specification sets forth that gradient stops are included in the SVG 'color-interpolation' property. As such, reference Steele intrinsically teaches this limitation.

As to claim 9,

The method of claim 6 wherein receiving brush data comprises receiving data corresponding a radial gradient brush.

Beda and Louveaux do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets forth that a user can paint with a

gradient that can be radial and also see sections 11.7.1 and 11.7.2 for more detail, thus meeting this limitation.

As to claim 10,

The method of claim 6 wherein receiving brush data comprises receiving data corresponding to an image.

Beda and Louveaux do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets forth that a user can paint with an image with further details provided in section 11.7.5 under the 'image-rendering' property.

As to claim 11,

The method of claim 10 further comprising, receiving markup corresponding to an image effect to apply to the image.

Beda and Louveaux do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 14.4 of the SVG specification sets forth that a user can use any image as an opacity mask, thus meeting this limitation, given that alpha blending is prima facie an image effect. The rendering functionality is inherent in SVG - see section 11.7, 14.4, et cetera.

As to claim 12,

The method of claim 1 further comprising, receiving pen data in association with the function call, and wherein causing data to be modified comprises invoking a pen function that defines an outline of a shape.

Beda and Louveaux do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 1 and reference SVG clearly supports this position. The term 'pen data' used by applicant above is comparable or analogous to any set of data defining the outline of a shape, including SVG 'path' data. Section 11.3 of the SVG specification sets forth that a user can fill a path that would correspond to the outline of shape with multiple illustrations provided for this under the 'nonzero' and 'even odd' subheadings - see details on paths -- with further details provided in section 11.3 and 11.4 (the individual strokes that create these effects.

As to claim 13,

The method of claim 1 wherein causing the data in a scene graph at least one of the set containing rectangle, polyline, polygon, path, line and ellipse shapes.

Beda and Louveaux do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 1 and reference SVG clearly supports this position. The SVG specification sets forth classes of shapes in section 9.1, where all six of the recited shapes (rectangle, polygon, path, line, polyline, and ellipse) are set forth. Further, the SVG view in Steele decomposes SVG instructions into scene graphs containing basic SVG shapes as above [Steele 0052], where 'Visual Elements' include

Shape classes. SVG is a markup language, therefore any SVG rendering utility would prima facie receive markup.

As to claim 14,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a geometry-related function to represent a rectangle in the scene graph data structure.

Beda and Louveaux do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. The SVG specification clearly shows in section 9.1 that rectangles are a basic shape, and further that in 9.2 under Example rect02 that such rectangles can have rounded corners, and code is provided that implements such. Also, the 'Rect' class inherently has geometry-related functions as set forth in the beginning to section 9.2. SVG is a markup language, therefore any SVG rendering utility would prima facie receive markup. As such, reference Steele shows a rectangle 715 in the scene graph in Figure 7 that intrinsically teaches this limitation. Also, said element can be animated under SVG section 19.2. Steele clearly teaches data modification in [0061] as set forth above.

As to claim 15,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a geometry-related function to represent a path in the scene graph data structure.

Beda and Louveaux do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Further, Steele Fig. 6 shows an animation where path information is extracted into element 620, and listed in Fig. 7 [see Steele 0050 and 0079]. Further, the SVG specification sets forth path data in section 9.1 as existing and how a 'path' can define a shape or similar. Both meanings are covered herein. Steele clearly teaches data modification in [0061] as set forth above.

Also, said element can be animated under SVG section 19.2.

As to claim 16,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a geometry-related function to represent a line in the scene graph data structure.

Beda and Louveaux do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Further, Steele Fig. 6 shows an animation where path information is extracted into element 620, and listed in Fig. 7 [see Steele 0050 and 0079]. A line element can be animated under SVG section 19.2, which is obviously geometric. Line elements are set forth in SVG section 9.5, and their geometric functions. Steele clearly teaches data modification in [0061] as set forth above.

The scene graph shown in Figure 7 could clearly include lines since they are Visual Elements [Steele 0060-0061, which supports animation, et cetera].

As to claim 17,

The method of claim 1 wherein the markup is related to hit-testing a visual in the scene graph data structure.

Beda and Louveaux do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Clearly, Steele teaches or implies navigation in [0067, 0085] - that is, navigation using a UI through a two-dimensional view, which is what any display normally shows. Therefore, given that a portable computer could clearly be used, the user would clearly be interacting with the display. As such, hit testing would be required for user interactivity, as could the system of Steele under the same rationale. The SVG specification sets forth hit testing in section 16.6 (the two paragraphs right at the end of the section) where hit testing (namely, hit detection) is taught, specifically testing text for character or cell hits and testing visual elements for hits in their entirety, and such information is clearly communicated in markup language - see section 16.2 for event types and elements transmitted in markup, for example. Steele clearly teaches data modification in [0061] as set forth above.

As to claim 18,

The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a function related to transforming coordinates of a visual in the scene graph data structure.

Beda and Louveaux do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Further, Steele Fig. 6 shows an animation where path information is extracted

into element 620, and listed in Fig. 7 [see Steele 0050 and 0079]. Clearly, SVG teaches the animation of visual elements, see section 19.2, which prima facie involves transforming coordinates of a visual in the scene graph data, and according to Steele [0052-0053] and a tree of elements can also be transformed [Steele 0052]. Steele , clearly teaches data modification in [0061] as set forth above. The scene graph shown in Figure 7 could clearly include lines since they are Visual Elements [Steele 0060-0061, which supports animation, et cetera].

As to claim 19,

The method of claim 1 wherein the markup is related to calculating a bounding box of a visual in the scene graph data structure.

Beda and Louveaux do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Bounding box calculations are taught in section 7.1 and detailed in section 7.11 where they are calculated. Steele clearly teaches data modification in [0061] as set forth above.

As to claim 20,

The method of claim 1 wherein causing data in the scene graph be modified comprises invoking a function via a common interface to a visual object in the scene graph data structure.

Beda and Louveaux do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Clearly, the SVG specification teaches interfaces in section 4.3 - there are

common DOM interface as set forth there. If the intended meaning of applicant was that such interfaces were based in hardware or software, fundamentally in reference Steele the user interacts with the browser that would provide a common interface, in that all events generated by such browser would go to an interface - that is, Steele clearly sets forth that his invention has various possible interfaces, depending on the embodiment (e.g. PDA, cell phone, et cetera [0004] and [0007-0008]). Steele clearly teaches data modification in [0061] as set forth above.

As to claim 21,

The method of claim 1 further comprising invoking a visual manager to render a tree of at least one visual object to a rendering target.

Beda and Louveaux do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Further, Steele Fig. 6 shows an animation where path information is extracted into element 620, and listed in Fig. 7 [see Steele 0050 and 0079] as trees. SVG teaches that all implementations must implement a rendering model as set forth in 3.1 and so forth, and scene graphs are known to directed acyclic, i.e. trees. Clearly, this model is implemented through the DOM interfaces set forth in section 4, and each element has its own element information that controls rendering aspects. Steele clearly teaches data modification in [0061] as set forth above. It is prima facie obvious that a 'visual manager' of some form must exist in order to handle formatting issues and precedence in rendering, and Steele teaches such a manager in [0075-0076. Clearly the rendering

information each visual element [Steele 0056-0061] is sufficient such that it is its own 'rendering target' as set forth above.

As to claim 22,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place a container object in the scene graph data structure, the contained object configured to contain at least one visual object.

Beda and Louveaux do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Further, Steele Fig. 7 shows a tree derived from an animation is shown - Figure 6 [see Steele 0050 and 0079]. SVG clearly teaches the use of container objects, as in section 1.6 it clearly teaches the use of 'container elements', which are defined as 'An element that can have graphic elements and other container elements as child elements'. Steele clearly teaches data modification in [0061] as set forth above. Clearly, the container object could be the head object of the tree structure shown in Steele Fig. 7.

As to claim 23,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place image data into the scene graph data structure.

Beda and Louveaux do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this

position. Clearly, visual elements can be covered by or tiled with images as established in SVG section 11.1, where SVG teaches: "...can paint (i.e. fill or stroke) with: ...a pattern (vector or image, possibly tiled) ..." which clearly establishes this, with more detail in section 11.7.5 and 11.12.

As to claim 24,

The method of claim 23 wherein causing data in the scene graph to be modified comprises invoking a function to place an image effect object into the scene graph data structure that is associated with the image data.

Beda and Louveaux do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 14.4 of the SVG specification sets forth that a user can use any image as an opacity mask for any visual element, thus meeting this limitation, given that alpha blending is prima facie an image effect. Steele clearly teaches data modification in [0061] as set forth above.

As to claim 25,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place data corresponding to text into the scene graph data structure.

Beda and Louveaux do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 10.1 of the SVG specification sets forth the use of a 'text' element, and

Steele teaches the inclusion of text element 725 in the data tree shown in Fig. 7. Steele clearly teaches data modification in [0061] as set forth above.

As to claim 26,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to provide a drawing context in response to the function call.

Beda and Louveaux do not expressly teach, but reference Steele teaches this limitation, as for example he teaches the retrieval of device context in [0101]. Clearly, the device receives information based on its device context, which clearly is associate with the drawing context, as the two are one and the same in this case. For the second case, if the definition of context is the data associated with a specific element - e.g. the details of the element, its location, color, et cetera, these attributes are inherent in SVG elements as set forth in the rejections above, e.g. sections 11.1, 9.1-9.7, et cetera. Further, Steele teaches the same in Figure 7, where each element has certain properties that would be a drawing context, in the sense that each visual element has those properties associated with it [Steele 0052-0056 and 0059-0061]. SVG is also a subset of XML, and SVG teaches metadata use in section 21.1. Steele clearly teaches data modification in [0061] as set forth above.

It further would have been obvious to utilize a device specific context so as to optimize data streamed to that device for purposes of minimizing memory consumption (a large problem pointed out by Steele [0007]), and the SVG DOM interfaces in section

4.1-4.4 (SVG specification) clearly provide methods for retrieving drawing information, which would be that context.

As to claim 27,

The method of claim 26 wherein the function call corresponds to a retained visual, and further comprising, calling back to have the drawing context of the retained visual returned to the scene graph data structure.

Beda and Louveaux do not expressly teach, but reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification clearly sets forth that all elements (as well as 3.1 and 4.2) have properties associated with them. The system of Steele clearly caches visuals during processing - see [0083], and it would be obvious that such data would be pulled from the cache to find out the state and properties of a visual element. Steele clearly teaches data modification in [0061] as set forth above. Further, it would be obvious to one of ordinary skill to cache the visuals so that they would be retained and that data would be pulled from the cache as set forth above, and as the Steele reference sets forth to have it pulled from there during data processing, including that of data trees like unto the one in Figure 7, as in [0100 Steele].

As to claim 30,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place animation data into the scene graph data structure.

Beda and Louveaux do not expressly teach, but reference Steele does teach it. Steele in Figs. 6 and 9 shows animated elements [0041] and in Fig. 7 shows that each subgroup is shifted a certain amount with x and y coordinates given. Steele [0050, 0052] for example provides that such animation takes place, and the SVG standard in 19.1 - 19.5 clearly sets forth how each element can have animation associated with it, which clearly is placed into the scene graph of Fig. 7. Therefore, clearly animation data is put into the tree of Fig. 7 Steele, which is clearly a scene graph by every known definition of the term, and a sample SVG XML program is provided in the second page of Fig. 9.

As to claim 31,

The method of claim 30 further comprising communicating timeline information corresponding to the animation data to a composition engine.

Beda and Louveaux do not expressly teach, but reference Steele clearly establishes in [0051-0054] and Figs. 6 and 9 that animation takes place through the SVG standard. Section 19.2 of SVG sets forth how this takes place, and at the bottom three paragraphs of section 19.2.2 it clearly states that animation has a document start and document end, and further in the second to last paragraph that the SVG system indicates the timeline position of document fragments being animated. Further, according to SVG 19.2.2 the animation is by document fragment and object path, which clearly are passed to the system is specified in, for example, the second page of Fig. 9 in the SVG .XML program. Clearly, the system of Steele performs compositing and rendering [0007, 0011-0012]. Finally, reference SVG teaches that it supports

compositing (section 14.2.1). The composition engine would be, for example, the composition engine of Steele, or the like.

As to claim 32,

The method of claim 31 wherein the composition engine interpolates graphics data based on the timeline to animate an output corresponding to an object in the scene graph data structure.

Beda and Louveaux do not expressly teach, but reference Steele does teach it. Steele in Figs. 6 and 9 shows animated elements [0041] and in Fig. 8, clearly during an animation the actions are shown, where the system in steps 835, 840, and 845 performs interpolation for nodes shown in the tree in Fig. 7. Clearly interpolation takes place during animation [0072, 0077-0079] which performs interpolation during the animation process as set forth in the SVG standard, and prima facie the output would only be objects in the scene graph, and they would prima facie be based on the timeline as set forth in the rejection to claim 31 above.

As to claim 33,

The method of claim 1 wherein receiving a function call via an interface of a media integration layer comprises receiving markup, and wherein causing data in a scene graph to be modified comprises parsing the markup into a all to an interface of an object.

Beda and Louveaux do not expressly teach, however, Steele clearly teaches the use of a SVG DOM as element 305 in Figure 3, where that intermediate format is then transferred to the BF object model 315, where this clearly represents a scene graph

(see Figure 6). SVG is a form of XML, e.g. XML-based markup. The XML must be interpreted to create the internal representation through the DOM such that it creates the representation where operations upon SVG objects can be made. Thus, Steele is clearly capable of receiving XML-based markup (see the rejection to claim 2 for further details, which is incorporated by reference in its entirety.

As to claim 34,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place an object corresponding to audio and/or video data into the scene graph data structure.

Beda and Louveaux do not expressly teach, but reference Steele does teach it. Steele in Figs. 8 shows audio elements 820 and 830 in the animation execution and in Fig. 7 a scene graph data structure (a tree). Steele [0050, 0052] for example provides that such animation takes place, and the SVG standard in 6.18 clearly sets forth aural style sheets, that are audio data that each element can have animation associated with it, which clearly is placed into the scene graph of Fig. 7. Also, by definition, SVG animations would be video.

As to claim 35,

The method of claim 1 wherein causing data in the scene graph to be modified comprises changing a mutable value of an object in the scene graph data structure.

Beda and Louveaux do not expressly teach, but reference Steele does teach it. Steele teaches in [0014] that one embodiment of his invention changes the visual graph

in accordance to changes of the sequence graph, where the visual graph is comparable to the "scene graph" of applicant and mutable values (e.g. position) of elements in the tree are shifted as per Steele [0052-0057]. Therefore, the limitation is met, and it would have been obvious to modify it so that it in fact change mutable values on the tree if applicant feels that this is not an adequate traversal of this particular limitation.

Claims 28-29 are rejected under 35 USC 103(a) as unpatentable over Beda, Louveaux, Steele and SVG and further in view of Kim et al (US 2003/0120823; hereinafter Kim).

As to claim 28,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place a three-dimensional visual into the scene graph data structure.

Beda, Louveaux, Steele and SVG teach causing data in a scene graph data structure to be modified comprises invoking a function. Steele clearly teaches the use of a SVG DOM as element 305 in Figure 3, where that intermediate format is then transferred to the BF object model 315, where this clearly represents a scene graph (see Figure 6). SVG is a form of XML, e.g. XML-based markup. The XML must be interpreted to create the internal representation through the DOM such that it creates the representation where operations upon SVG objects can be made. Thus, Steele is clearly capable of receiving XML-based markup.

However, they do not teach invoking a function to place a three-dimensional visual into the scene graph data structure. The Kim reference clearly teaches this limitation. The Kim reference clearly teaches scene graphs as established in the rejection to claim 1 [0007-0009]. Clearly, Kim teaches the use of three-dimensional data under the X3D standard specification, which is a form of XML [0007-0009]. Clearly, Kim teaches [0032-0034] that scene data is processed and all objects have specific sets of data associated with them, for example [0042-0044], which clearly establishes that all objects have three-dimensional attributes and properties. This prima facie establishes that three-dimensional visuals are placed into the scene graph data structure. Clearly, the system implements the X3D specification in software, and, as such, it is software, which prima facie uses function calls.

Kim [0007-0008] clearly teaches the use of a scene graph and that X3D requires the construction of such scene graphs from primitives. Kim further teaches that the user can move through a scene [0020, 0026], which clearly establishes that a user is navigating and the scene is constantly being re-rendered, which prima facie requires data in the scene graph to be modified.

Kim extols the benefits of three-dimensional graphics in [0001-0007]. Therefore, based on the above teachings, it would have been obvious to one of ordinary skill in the art at the time the invention was made to modify Beda, Louveaux, Steele and SVG to have three-dimensional elements.

As to claim 29,

The method of claim 28 wherein causing data in the scene graph to be modified comprises mapping a two-dimensional surface onto the three dimensional visual.

Beda, Louveaux, Steele and SVG fail to expressly teach, but the Kim reference clearly teaches this limitation, and X3D standard is only cited to clarify certain points. The Kim reference clearly teaches scene graphs as established in the rejection to claim 1 [0007- 0009]. Clearly, Kim teaches the use of three-dimensional data under the X3D standard specification, which is a form of XML [0007-0009]. Clearly, Kim teaches [0032-0034] that scene data is processed and all objects have specific sets of data associated with them, for example [0042-0044], which clearly establishes that all objects have three- dimensional attributes and properties. This prima facie establishes that three-dimensional visuals are placed into the scene graph data structure. Clearly, the system implements the X3D specification in software, and, as such, it is software, which prima facie uses function calls. Secondly, the X3D standard clearly allows for the incorporation of 2D images onto three-dimensional elements, as stated in X3D 18.2.1 and 18.4.1, parti(~ularly 18.4.1, which reads clearly that "browsers may support other image formats ... which may be rendered into a 2D image" and clearly those images can be applied to three-dimensional objects such as those described in 18.3.1 and as defined in 18.2.1-18.2.3. Motivation and rationale are taken from the rejection to claim 28 above.

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to JWALANT AMIN whose telephone number is (571)272-2455. The examiner can normally be reached on 9:30 a.m. - 6:00 p.m..

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Mark Zimmerman can be reached on 571-272-7653. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/J. A./
Examiner, Art Unit 2628

/Mark K Zimmerman/
Supervisory Patent Examiner, Art Unit 2628